1320 Bundling

Outel, a famous semiconductor company, released recently a new model of microprocessor called Platinium. Like many modern processors, Platinium can execute many instructions in one clock step providing that there are no dependencies between them (instruction I_2 is dependent on instruction I_1 if for example I_2 reads a register that I_1 writes to). Some processors are so clever that they calculate on they which instructions can be safely executed in parallel. Platinium however expects this information to be explicitly specified. A special marker, called simply a *stop*, inserted between two instructions indicatesthat some instructions after the stop are possibly dependent on some instructions before the stop. In other words instructions between two successive stops can be executed in parallel and there should not be dependencies between them.

Another interesting feature of Platinium is that a sequence of instructions must be split into groups of one, two or three successive instructions. Each group has to be packed into a container called a *bundle*. Each bundle has 3 slots and a single instruction can be put into each slot, however some slots may stay empty. Each instruction is categorized into one of 10 instruction types denoted by consecutive capital letters from `A` to `J` (instructions of the same type have similar functionality, for example type `A` groups integer arithmetic instructions and type `F` groups floating-point instructions). Only instructions of certain types are allowed to be packed into one bundle. A *template* specifies one permissible combination of instruction types within a bundle. A template can also specify a position of a stop in the middle of a bundle (there is at most one such stop allowed). In addition, stops are allowed between any two adjoining bundles. A set of templates is called a *bundling profile*. When packing instructions into bundles, one has to use templates from bundling profile only.

Although Platinium is equipped with an instruction cache it was found that for maximal performance it is most crucial to pack instructions as densely as possible. Second important thing is to use a small number of stops.

Your task is to write a program for bundling Platinium instructions. For the sake of simplicity we assume that the instructions cannot be reordered.

Task

Write a program that:

- reads a bundling prole and a sequence of instructions,
- computes the minimal number of bundles into which the sequence can be packed without breaking the dependencies and the minimal number of all stops that are required for the minimal number of bundles,
- writes the result.

Input

Input consists of several test case, each separated by a blank line. The first line of the file indicates the number of test cases, and it's followed by a blank line.

The first line of each dataset contains two integers t and n separated by a single space. Integer t ($1 \leq t \leq 1500$) is the number of templates in the bundling profile. Integer n ($1 \leq n \leq 100000$) is the number of instructions to be bundled.

Each of the next t lines specifies one template and contains 3 capital letters t_1, t_2, t_3 with no spaces in between followed by a space and an integer p. Letter t_i (`A` $\leq t_i \leq$ `J`) is an instruction type allowed in

the i-th slot. Integer p ($0 \leq p \leq 2$) is the index of the slot after which the stop is positioned (0 means no stop within the bundle).

Each of the next n lines specifies one instruction. The i-th line of these n lines contains one capital letter c_i and an integer d_i, separated by a single space. Letter c_i (`A` $\leq c_i \leq$ `J`) is the type of the i-th instruction. Integer d_i ($0 \leq d_i < i$) is the index of the last instruction (among the previous ones) that the i-th instruction is dependent on (0 means that the instruction is not dependent on any former instruction).

You can assume that for each instruction type c appearing in the instruction sequence there is at least one template containing c.

Output

The first and only line of the output contains two integers b and s separated by a single space. Integer b is the minimal number of bundles in a valid packing. Integer s is the minimal number of all stops that are required for the minimal number of bundles.

Print a blank line between test cases.

Sample Input

```
1

4 9
ABB 0
BAD 1
AAB 0
ABB 2
B 0
B 1
A 1
A 1
B 4
D 0
A 0
B 3
B 0
```

Sample Output

```
4 3
```